Quantum

StorNext 3.1®

StorNext

6-01376-07

StorNext 3.1 File System Tuning Guide, 6-01376-07, Ver. A, Rel. 3.1, October 2007, Made in USA.

Quantum Corporation provides this publication "as is" without warranty of any kind, either express or implied, including but not limited to the implied warranties of merchantability or fitness for a particular purpose. Quantum Corporation may revise this publication from time to time without notice.

COPYRIGHT STATEMENT

Copyright 2007 by Quantum Corporation. All rights reserved.

StorNext copyright (c) 1991-2007 Advanced Digital Information Corporation (ADIC), Redmond, WA, USA. All rights reserved.

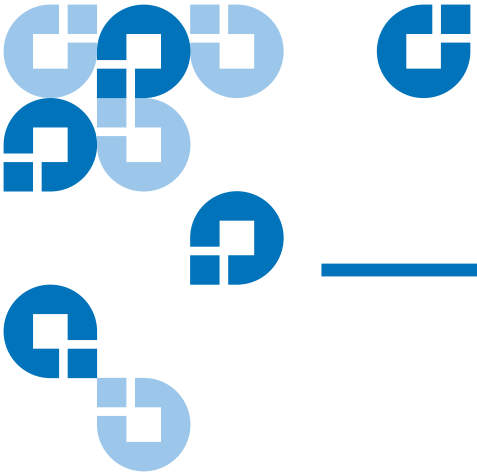
Your right to copy this manual is limited by copyright law. Making copies or adaptations without prior written authorization of Quantum Corporation is prohibited by law and constitutes a punishable violation of the law.

TRADEMARK STATEMENT

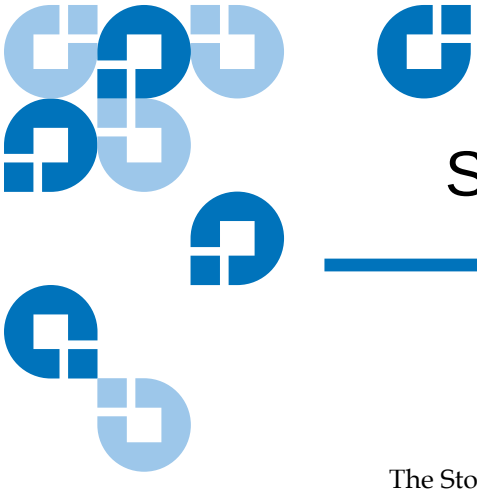
Quantum, DLT, DLTtape, the Quantum logo, and the DLTtape logo are all registered trademarks of Quantum Corporation.

SDLT and Super DLTtape are trademarks of Quantum Corporation.

Other trademarks may be mentioned herein which belong to other companies.



Contents



StorNext File System Tuning

The StorNext File System (SNFS) provides extremely high performance for widely varying scenarios. Many factors determine the level of performance you will realize. In particular, the performance characteristics of the underlying storage system are the most critical factors. However, other components such as the Metadata Network and MDC systems also have a significant effect on performance.

Furthermore, file size mix and application I/O characteristics may also present specific performance requirements, so SNFS provides a wide variety of tunable settings to achieve optimal performance. It is usually best to use the default SNFS settings, because these are designed to provide optimal performance under most scenarios. However, this guide discusses circumstances in which special settings may offer a performance benefit.

The Underlying Storage System

The performance characteristics of the underlying storage system are the most critical factors for file system performance. Typically, RAID storage systems provide many tuning options for cache settings, RAID level, segment size, stripe size, and so on.

RAID Cache Configuration

The single most important RAID tuning component is the cache configuration. This is particularly true for small I/O operations. Contemporary RAID systems such as the EMC CX series and the various Engenio systems provide excellent small I/O performance with properly tuned caching. So, for the best general purpose performance characteristics, it is crucial to utilize the RAID system caching as fully as possible.

For example, write-back caching is absolutely essential for metadata stripe groups to achieve high metadata operations throughput.

However, there are a few drawbacks to consider as well. For example, read-ahead caching improves sequential read performance but might reduce random performance. Write-back caching is critical for small write performance but may limit peak large I/O throughput. Some RAID systems cannot safely support write-back caching without risk of data loss, which is not suitable for critical data such as file system metadata.

Consequently, this is an area that requires an understanding of application I/O requirements. As a general rule, RAID system caching is critically important for most applications, so it is the first place to focus tuning attention.

RAIDWrite-BackCaching

Write-back caching dramatically reduces latency in small write operations. This is accomplished by returning a successful reply as soon as data is written into cache, and then deferring the operation of actually writing the data to the physical disks. This results in a great performance improvement for small I/O operations.

Many contemporary RAID systems protect against write-back cache data loss due to power or component failure. This is accomplished through various techniques including redundancy, battery backup, battery-backed memory, and controller mirroring. To prevent data corruption, it is important to ensure that these systems are working properly. It is particularly catastrophic if file system metadata is corrupted, because complete file system loss could result. Check with your RAID vendor to make sure that write-back caching is safe to use.

Minimal I/O latency is critically important for metadata stripe groups to achieve high metadata operations throughput. This is because metadata operations involve a very high rate of small writes to the metadata disk, so disk latency is the critical performance factor. Write-back caching can be an effective approach to minimizing I/O latency and optimizing metadata operations throughput. This is easily observed in the hourly

File System Manager (FSM) statistics reports in the **cvlog** file. For example, here is a message line from the cvlog file:

```
PIO HiPriWr SUMMARY SnmsMetaDisk0 sysavg/350 sysmin/333 sysmax/367
```

This statistics message reports average, minimum, and maximum write latency (in microseconds) for the reporting period. If the observed average latency exceeds 500 microseconds, peak metadata operation throughput will be degraded. For example, create operations may be around 2000 per second when metadata disk latency is below 500 microseconds. However, if metadata disk latency is around 5 milliseconds, create operations per second may be degraded to 200 or worse.

Another typical write caching approach is a "write-through." This approach involves synchronous writes to the physical disk before returning a successful reply for the I/O operation. The write-through approach exhibits much worse latency than write-back caching; therefore, small I/O performance (such as metadata operations) is severely impacted. It is important to determine which write caching approach is employed, because the performance observed will differ greatly for small write I/O operations.

In some cases, large write I/O operations can also benefit from caching. However, some SNFS customers observe maximum large I/O throughput by disabling caching. While this may be beneficial for special large I/O scenarios, it severely degrades small I/O performance; therefore, it is suboptimal for general-purpose file system performance.

RAID Read-Ahead Caching

RAID read-ahead caching is a very effective way to improve sequential read performance for both small (buffered) and large (DMA) I/O operations. When this setting is utilized, the RAID controller pre-fetches disk blocks for sequential read operations. Therefore, subsequent application read operations benefit from cache speed throughput, which is faster than the physical disk throughput.

This is particularly important for concurrent file streams and mixed I/O streams, because read-ahead significantly reduces disk head movement that otherwise severely impacts performance.

While read-ahead caching improves sequential read performance, it does not help highly transactional performance. Furthermore, some SNFS customers actually observe maximum large sequential read throughput by disabling caching. While disabling read-ahead is beneficial in these

unusual cases, it severely degrades typical scenarios. Therefore, it is unsuitable for most environments.

RAID Level, Segment Size, and Stripe Size

Configuration settings such as RAID level, segment size, and stripe size are very important and cannot be changed after put into production, so it is critical to determine appropriate settings during initial configuration.

The best RAID level to use for high I/O throughput is usually RAID5. The stripe size is determined by the product of the number of disks in the RAID group and the segment size. For example, a 4+1 RAID5 group with 64K segment size results in a 256K stripe size. The stripe size is a very critical factor for write performance because I/Os smaller than the stripe size may incur a read/modify/write penalty. It is best to configure RAID5 settings with no more than 512K stripe size to avoid the read/modify/write penalty. The read/modify/write penalty is most noticeable in the absence of "write-back" caching being performed by the RAID controller.

The RAID stripe size configuration should typically match the SNFS **StripeBreadth** configuration setting when multiple LUNs are utilized in a stripe group. However, in some cases it might be optimal to configure the SNFS **StripeBreadth** as a multiple of the RAID stripe size, such as when the RAID stripe size is small but the user's I/O sizes are very large. However, this will be suboptimal for small I/O performance, so may not be suitable for general purpose usage.

RAID1 mirroring is the best RAID level for metadata and journal storage because it is most optimal for very small I/O sizes. It is also very important to allocate entire physical disks for the Metadata and Journal LUNs in ordep to avoid bandwidth contention with other I/O traffic. Metadata and Journal storage requires very high IOPS rates (low latency) for optimal performance, so contention can severely impact IOPS (and latency) and thus overall performance. If Journal I/O exceeds 1ms average latency, you will observe significant performance degradation.

It can be useful to use a tool such as **lmdd** to help determine the storage system performance characteristics and choose optimal settings. For example, varying the stripe size and running lmdd with a range of I/O sizes might be useful to determine an optimal stripe size multiple to configure the SNFS **StripeBreadth**.

File Size Mix and Application I/O Characteristics

It is always valuable to understand the file size mix of the target dataset as well as the application I/O characteristics. This includes the number of concurrent streams, proportion of read versus write streams, I/O size, sequential versus random, Network File System (NFS) or Common Internet File System (CIFS) access, and so on.

For example, if the dataset is dominated by small or large files, various settings can be optimized for the target size range.

Similarly, it might be beneficial to optimize for particular application I/O characteristics. For example, to optimize for sequential 1MB I/O size it would be beneficial to configure a stripe group with four 4+1 RAID5 LUNs with 256K stripe size.

However, optimizing for random I/O performance can incur a performance trade-off with sequential I/O.

Furthermore, NFS and CIFS access have special requirements to consider as described in the Direct Memory Access (DMA) I/O Transfer section.

Direct Memory Access (DMA) I/O Transfer

To achieve the highest possible large sequential I/O transfer throughput, SNFS provides DMA-based I/O. To utilize DMA I/O, the application must issue its reads and writes of sufficient size and alignment. This is called well-formed I/O. See the **mount command** settings **auto_dma_read_length** and **auto_dma_write_length**, described in the Mount Command Options on page 17.

Buffer Cache

Reads and writes that aren't well-formed utilize the SNFS buffer cache. This also includes NFS or CIFS-based traffic because the NFS and CIFS daemons defeat well-formed I/Os issued by the application.

There are several configuration parameters that affect buffer cache performance. The most critical is the RAID cache configuration because buffered I/O is usually smaller than the RAID stripe size, and therefore incurs a read/modify/write penalty. It might also be possible to match the RAID stripe size to the buffer cache I/O size. However, it is typically most important to optimize the RAID cache configuration settings described earlier in this document.

It is usually best to configure the RAID stripe size no greater than 256K for optimal small file buffer cache performance.

For more buffer cache configuration settings, see Mount Command Options on page 17.

NFS / CIFS

It is best to isolate NFS and/or CIFS traffic off of the metadata network to eliminate contention that will impact performance. For optimal performance it is necessary to use 1000BaseT instead of 100BaseT. On NFS clients, use the **vers=3**, **rsize=262144** and **wsize=262144** mount options, and use TCP mounts instead of UDP. When possible, it is also best to utilize TCP Offload capabilities as well as jumbo frames.

It is best practice to have clients directly attached to the same network switch as the NFS or CIFS server. Any routing required for NFS or CIFS traffic incurs additional latency that impacts performance.

It is critical to make sure the **speed/duplex** settings are correct, because this severely impacts performance. Most of the time auto-detect is the correct setting. Some managed switches allow setting **speed/duplex** (for example 1000Mb/full,) which disables **auto-detect** and requires the host to be set exactly the same. However, if the settings do not match between switch and host, it severely impacts performance. For example, if the switch is set to auto-detect but the host is set to 1000Mb/full, you will observe a high error rate along with extremely poor performance. On Linux, the **mii-diag** tool can be very useful to investigate and adjust **speed/duplex** settings.

If performance requirements cannot be achieved with NFS or CIFS, consider using a StorNext Distributed LAN client or fibre-channel attached client.

It can be useful to use a tool such as **netperf** to help verify network performance characteristics.

The Metadata Network

As with any client/server protocol, SNFS performance is subject to the limitations of the underlying network. Therefore, it is recommended that you use a dedicated Metadata Network to avoid contention with other network traffic. Either 100BaseT or 1000BaseT is required, but for a dedicated Metadata Network there is usually no benefit from using 1000BaseT over 100BaseT. Neither TCP offload nor are jumbo frames required.

It is best practice to have all SNFS clients directly attached to the same network switch as the MDC systems. Any routing required for metadata traffic will incur additional latency that impacts performance.

It is critical to ensure that **speed/duplex** settings are correct, as this will severely impact performance. Most of the time **auto-detect** is the correct setting. Some managed switches allow setting **speed/duplex**, such as **100Mb/full**, which disables **auto-detect** and requires the host to be set exactly the same. However, performance is severely impacted if the settings do not match between switch and host. For example, if the switch is set to **auto-detect** but the host is set to **100Mb/full**, you will observe a high error rate and extremely poor performance. On Linux the **mii-diag** tool can be very useful to investigate and adjust **speed/duplex** settings.

It can be useful to use a tool like **netperf** to help verify the Metadata Network performance characteristics. For example, if **netperf -t TCP_RR** reports less than 15,000 transactions per second capacity, a performance penalty may be incurred.

The Metadata Controller System

The CPU power and memory capacity of the MDC System are important performance factors, as well as the number of file systems hosted per system. In order to ensure fast response time it is necessary to use dedicated systems, limit the number of file systems hosted per system (maximum 8), and have an adequate CPU and memory.

Some metadata operations such as file creation can be CPU intensive, and benefit from increased CPU power. The MDC platform is important in these scenarios because lower clock- speed CPUs such as Sparc and Mips degrade performance.

Other operations can benefit greatly from increased memory, such as directory traversal. SNFS provides three config file settings that can be used to realize performance gains from increased memory: **BufferCacheSize**, **InodeCacheSize**, and **ThreadPoolSize**.

However, it is critical that the MDC system have enough physical memory available to ensure that the FSM process doesn't get swapped out. Otherwise, severe performance degradation and system instability can result.

FSM Configuration File Settings

The following FSM configuration file settings are explained in greater detail in the **cvfs_config man** page. For a sample FSM configuration file, see Sample FSM Configuration File on page 25.

The examples in the following sections are excerpted from the sample configuration file from Sample FSM Configuration File on page 25.

Stripe Groups

Splitting apart data, metadata, and journal into separate stripe groups is usually the most important performance tactic. The **create**, **remove**, and **allocate** (e.g., write) operations are very sensitive to I/O latency of the journal stripe group. Configuring a separate stripe group for journal greatly benefits the speed of these operations because disk seek latency is minimized. However, if **create**, **remove**, and **allocate** performance aren't critical, it is okay to share a stripe group for both metadata and journal, but be sure to set the exclusive property on the stripe group so it doesn't get allocated for data as well. It is recommended that you assign only a single LUN for each journal or metadata stripe group. Multiple metadata stripe groups can be utilized to increase metadata I/O throughput through concurrency. RAID1 mirroring is optimal for metadata and journal storage. Utilizing the write-back caching feature of the RAID system (as described previously) is critical to optimizing performance of the journal and metadata stripe groups.

Example:

```
[stripeGroup RegularFiles]
Status UP
Exclusive No          ##Non-Exclusive stripeGroup for all Files##
Read Enabled
Write Enabled
StripeBreadth 256K
MultiPathMethod Rotate
Node CvfsDisk6 0
Node CvfsDisk7 1
```

Affinities

Affinities are another stripe group feature that can be very beneficial. Affinities can direct file allocation to appropriate stripe groups according to performance requirements. For example, stripe groups can be set up with unique hardware characteristics such as fast disk versus slow disk, or wide stripe versus narrow stripe. Affinities can then be employed to steer files to the appropriate stripe group.

For optimal performance, files that are accessed using large DMA-based I/O could be steered to wide-stripe stripe groups. Less performance-critical files could be steered to slow disk stripe groups. Small files could be steered to narrow-stripe stripe groups.

Example:

```
[stripeGroup VideoFiles]
Status UP
Exclusive Yes         ##These Two lines set Exclusive stripeGroup##
Affinity VideoFiles   ##for Video Files Only##
Read Enabled
Write Enabled
StripeBreadth 4M
MultiPathMethod Rotate
Node CvfsDisk2 0
Node CvfsDisk3 1
```

StripeBreadth

This setting must match the RAID stripe size or be a multiple of the RAID stripe size. Matching the RAID stripe size is usually the most optimal setting. However, depending on the RAID performance characteristics and application I/O size, it might be beneficial to use a multiple of the RAID stripe size. For example, if the RAID stripe size is 256K, the stripe group contains 4 LUNs, and the application to be optimized uses DMA I/O with 8MB block size, a **StripeBreadth** setting of 2MB might be optimal. In this example the 8MB application I/O is issued as 4 concurrent 2MB I/Os to the RAID. This concurrency can provide up to a 4X performance increase. This typically requires some experimentation to determine the RAID characteristics. The **lmdd** utility can be very helpful. Note that this setting is not adjustable after initial file system creation.

Example:

```
[stripeGroup AudioFiles]
Status UP
Exclusive Yes          ##These two lines set Exclusive stripeGroup ##
Affinity AudioFiles    ##for Audio Files Only##
Read Enabled
Write Enabled
StripeBreadth 1M
MultiPathMethod Rotate
Node CvfsDisk4 0
Node CvfsDisk5 1
```

BufferCacheSize

This setting consumes up to 2X bytes of memory times the number specified. Increasing this value can reduce latency of any metadata operation by performing a **hot cache** access to directory blocks, inode information, and other metadata info. This is about 10 to 1000 times faster than I/O. It is especially important to increase this setting if metadata I/O latency is high, (for example, more than 2ms average latency). We recommend sizing this according to how much memory is available; more is better.

Example: `# BufferCacheSize 64M # default 32MB`

InodeCacheSize

This setting consumes about 800-1000 bytes of memory times the number specified. Increasing this value can reduce latency of any metadata operation by performing a hot cache access to inode information instead of an I/O to get inode info from disk, about 100 to 1000 times faster. It is especially important to increase this setting if metadata I/O latency is high, (for example, more than 2ms average latency). You should try to size this according to the sum number of working set files for all clients. The recommended range is 16K to 64K.

Example: InodeCacheSize 16K # 800-1000 bytes each, default 8K

ThreadPoolSize

This setting consumes 512 KB memory times the number specified. Increasing this value can improve concurrency of metadata operations. For example, if many client processes are executing concurrently, the thread pool can become exhausted by I/O wait time. Increasing the thread pool size permits hot cache operations to be processed that would otherwise be backed up behind the I/O-bound operations. There are various O/S limits to the number of threads that can cause fatal problems for the FSM daemon, so it's not a good idea to set this setting too high. A range from 32 to 128 is recommended, depending on the amount of available memory. It is recommended to size it according to the **max threads** FSM hourly statistic reported in the **cvlog** file.

Example: ThreadPoolSize 32 # default 16, 512 KB memory per thread

ForcestripeAlignment

This setting should always be set to Yes. This is critical if the largest **StripeBreadth** defined is greater than 1MB. Note that this setting is not adjustable after initial file system creation.

Example: ForcestripeAlignment Yes

FsBlockSize

The optimal settings for both performance and space utilization are in the range of 16K, 32K, or 64K.

Settings greater than 64K are not recommended because performance will be adversely impacted due to inefficient metadata I/O operations. Values less than 16K are not recommended in most scenarios because startup and failover time may be adversely impacted. Setting FsBlockSize to higher values is important for multiterabyte file systems for optimal startup and failover time.

> **Note:** This is particularly true for slow CPU clock speed metadata servers such as Sparc. However, values greater than 16K can severely consume metadata space in cases where the file-to-directory ratio is low (e.g. less than 100 to 1).

The best rule of thumb is to use 16K unless other requirements such as directory ratio dictate otherwise.

This setting is not adjustable after initial file system creation, so it is very important to give it careful consideration during initial configuration.

Example: FsBlockSize 16K

JournalSize

The optimal settings are in the range between 16M and 64M. Avoid values greater than 64M due to potentially severe impacts on startup and failover times. Values at the higher end of the 16M-64M range may improve performance of metadata operations in some cases, although at the cost of slower startup and failover time. A good rule of thumb is to use 16M unless another requirement dictates differently. This setting is adjustable using the **cvupdatefs** utility. For more information, see the **cvupdatefs** man page.

Example: JournalSize 16M

SNFS Tools

The **snfsdefrag** tool is very useful to identify and correct file extent fragmentation. Reducing extent fragmentation can be very beneficial for performance. You can use this utility to determine whether files are fragmented, and if so, fix them. If your files are prone to fragmentation you should also use the FSM config file tuning options to minimize fragmentation. These global configuration settings are **InodeExpandMin**, **InodeExpandInc**, and **InodeExpandMax**. (For more information, see the

man **cvfs_config** page.) The **snfsdefrag** man page explains the command options in greater detail.

FSM hourly statistics reporting is another very useful tool. This can show you the mix of metadata operations being invoked by client processes, as well as latency information for metadata operations and metadata and journal I/O. This information is easily accessed in the cvlog log files. All of the latency oriented stats are reported in microsecond units.

It also possible to trigger an instant FSM statistics report by setting the **Once Only** debug flag using **cvadmin**. For example:

```
cvadmin -F snfs1 -e 'debug 0x01000000' ; tail -100 /usr/cvfs/data/snfs1/log/cvlog
```

The following items are a few things to watch out for:

- A non-zero value for **FSM wait SUMMARY journal waits** indicates insufficient IOPS performance of the disks assigned to the metadata stripe group. This usually requires reducing the metadata I/O latency time by adjusting RAID cache settings or reducing bandwidth contention for the metadata LUN. Another possible solution is to add another metadata stripe group to the file system. This will improve metadata ops performance through I/O concurrency.
- Non-zero value for **FSM wait SUMMARY free buffer waits** or low hit ratio for **FSM cache SUMMARY buffer lookups** indicates the FSM configuration setting **BufferCacheSize** is insufficient.
- Non-zero value for **FSM wait SUMMARY free inode waits** or low hit ratio for **FSM cache SUMMARY inode lookups** indicates the FSM configuration setting **InodeCacheSize** is insufficient.
- Large value for **FSM threads SUMMARY max busy** indicates the FSM configuration setting **ThreadPoolSize** is insufficient.
- Extremely high values for **FSM cache SUMMARY inode lookups**, **TKN SUMMARY TokenRequestV3**, or **TKN SUMMARY TokenReqAlloc** might indicate excessive file fragmentation. If so, the **snfsdefrag** utility can be used to fix the fragmented files.
- The **VOP** and **TKN** summary statistics indicate the count and avg/min/max microsecond latency for the various metadata operations. This shows what type of metadata operations are most prevalent and most costly. These are also broken out per client, which can be useful to identify a client that is disproportionately loading the FSM.

SNFS supports the Windows **Perfmon** utility. This provides many useful statistics counters for the SNFS client component. To install, obtain a copy of **cvfsperf.dll** from the SCM team in Denver and copy it into the c:/winnt/system32 directory on the SNFS client system. Then run **rmperfreg.exe** and **instperfreg.exe** to set up the required registry settings. After these steps, the SNFS counters should be visible to the **Windows Perfmon** utility. If not, check the Windows Application Event log for errors.

The **cvcp** utility is a higher performance alternative to commands such as **cp** and **tar**. The **cvcp** utility achieves high performance by using threads, large I/O buffers, preallocation, stripe alignment, DMA I/O transfer, and Bulk Create. Also, the **cvcp** utility uses the SNFS External API for preallocation and stripe alignment. In the directory-to-directory copy mode (for example, **cvcp source_dir destination_dir**,) **cvcp** conditionally uses the **Bulk Create** API to provide a dramatic small file copy performance boost. However, it will not use **Bulk Create** in some scenarios, such as non-root invocation, managed file systems, quotas, or Windows security. Hopefully, these limitations will be removed in a future release. When **Bulk Create** is utilized, it significantly boosts performance by reducing the number of metadata operations issued. For example, up to 20 files can be created all with a single metadata operation. For more information, see the **cvcp** man page.

The **cvmkfile** utility provides a command line tool to utilize valuable SNFS performance features. These features include preallocation, stripe alignment, and affinities. See the **cvmkfile** man page.

The **Lmdd** utility is very useful to measure raw LUN performance as well as varied I/O transfer sizes.

The **cvdbset** utility has a special "Perf" trace flag that is very useful to analyze I/O performance. For example: cvdbset perf

Then, you can use **cvdb -g** to collect trace information such as this:

PERF: Device Write 41 MB/s IOs 2 exts 1 offs 0x0 len 0x400000 mics 95589 ino 0x5

PERF: VFS Write EofDmaAlgn 41 MB/s offs 0x0 len 0x400000 mics 95618 ino 0x5

The "PERF: Device" trace shows throughput measured for the device I/O. It also shows the number of I/Os into which it was broken, and the number of extents (sequence of consecutive filesystem blocks).

The “PERF: VFS” trace shows throughput measured for the read or write system call and significant aspects of the I/O, including:

- Dma: DMA
- Buf: Buffered
- Eof: File extended
- Algn: Well-formed DMA I/O
- Shr: File is shared by another client
- Rt: File is real time
- Zr: Hole in file was zeroed

Both traces also report file offset, I/O size, latency (mics), and inode number.

Sample use cases:

- Verify that I/O properties are as expected.

  You can use the VFS trace to ensure that the displayed properties are consistent with expectations, such as being well formed; buffered versus DMA; shared/non-shared; or I/O size. If a small I/O is being performed DMA, performance will be poor. If DMA I/O is not well formed, it requires an extra data copy and may even be broken into small chunks. Zeroing holes in files has a performance impact.

- Determine if metadata operations are impacting performance.

  If VFS throughput is inconsistent or significantly less than Device throughput, it might be caused by metadata operations. In that case, it would be useful to display “fsmtoken,” “fsmvnops,” and “fsmdmig” traces in addition to “perf.”

- Identify disk performance issues.

  If Device throughput is inconsistent or less than expected, it might indicate a slow disk in a stripe group, or that RAID tuning is necessary.

- Identify file fragmentation.

  If the extent count “exts” is high, it might indicate a fragmentation problem.This causes the device I/Os to be broken into smaller chunks, which can significantly impact throughput.

- Identify read/modify/write condition.

  If buffered VFS writes are causing Device reads, it might be beneficial to match I/O request size to a multiple of the "cachebufsize" (default 64KB; see **mount_cvfs** man page). Another way to avoid this is by truncating the file before writing.

The **cvadmin** command includes a **latency-test** utility for measuring the latency between an FSM and one or more SNFS clients. This utility causes small messages to be exchanged between the FSM and clients as quickly as possible for a brief period of time, and reports the average time it took for each message to receive a response.

The **latency-test** command has the following syntax:

latency-test *index-number* [*seconds*]

latency-test all [*seconds*]

If an *index-number* is specified, the test is run between the currently-selected FSM and the specified client. (Client index numbers are displayed by the **cvadmin who** command). If **all** is specified, the test is run against each client in turn.

The test is run for 2 seconds, unless a value for seconds is specified.

Here is a sample run:

```
snadmin (lsi) > latency-test

Test started on client 1 (bigsky-node2)... latency 55us

Test started on client 2 (k4)... latency 163us
```

There is no rule-of-thumb for "good" or "bad" latency values. Latency can be affected by CPU load or SNFS load on either system, by unrelated Ethernet traffic, or other factors. However, for otherwise idle systems, differences in latency between different systems can indicate differences in hardware performance. (In the example above, the difference is a Gigabit Ethernet and faster CPU versus a 100BaseT Ethernet and a slower CPU.) Differences in latency over time for the same system can indicate new hardware problems, such as a network interface going bad.

If a latency test has been run for a particular client, the **cvadmin who long** command includes the test results in its output, along with information about when the test was last run.

Mount Command Options

The following SNFS mount command settings are explained in greater detail in the **mount_cvfs** man page.

The default size of the buffer cache varies by platform and main memory size, and ranges between 32MB and 256MB. And, by default, each buffer is 64K so the cache contains between 512 and 4096 buffers. In general, increasing the size of the buffer cache will not improve performance for streaming reads and writes. However, a large cache helps greatly in cases of multiple concurrent streams, and where files are being written and subsequently read. Buffer cache size is adjusted with the buffercachecap setting.

The buffer cache I/O size is adjusted using the cachebufsize setting. The default setting is usually optimal; however, sometimes performance can be improved by increasing this setting to match the RAID5 stripe size.

Using a large **cachebufsize** setting decreases random I/O performance when the amount of data being read is smaller than the cache buffer size.

Buffer cache read-ahead can be adjusted with the **buffercache_readahead** setting. When the system detects that a file is being read in its entirety, several buffer cache I/O daemons pre-fetch data from the file in the background for improved performance. The default setting is optimal in most scenarios.

The **auto_dma_read_length** and **auto_dma_write_length** settings determine the minimum transfer size where direct DMA I/O is performed instead of using the buffer cache for well-formed I/O. These settings can be useful when performance degradation is observed for small DMA I/O sizes compared to buffer cache.

For example, if buffer cache I/O throughput is 200 MB/sec but 512K DMA I/O size observes only 100MB/sec, it would be useful to determine which DMA I/O size matches the buffer cache performance and adjust **auto_dma_read_length** and **auto_dma_write_length** accordingly. The **lmdd** utility is handy here.

The **dircachesize** option sets the size of the directory information cache on the client. This cache can dramatically improve the speed of readdir operations by reducing metadata network message traffic between the SNFS client and FSM. Increasing this value improves performance in scenarios where very large directories are not observing the benefit of the client directory cache.

SNFS External API

The SNFS External API might be useful in some scenarios because it offers programmatic use of special SNFS performance capabilities such as affinities, preallocation, and quality of service. For more information, see the *Quality of Service* chapter of the *StorNext User's Guide API Guide*.

The Distributed LAN (Disk Proxy) Networks

As with any client/server protocol, SNFS Distributed LAN performance is subject to the limitations of the underlying network. Therefore, it is strongly recommended that you use Gigabit (1000BaseT) for Distributed LAN traffic. Neither TCP offload nor jumbo frames are required.

Hardware Configuration

SNFS Distributed LAN can easily fill several Gigabit Ethernets with data, so take special care when selecting and configuring the switches used to interconnect SNFS Distributed LAN clients and servers. Ensure that your network switches have enough internal bandwidth to handle all of the anticipated traffic between all Distributed LAN clients and servers connected to them.

A network switch that is dropping packets will cause TCP retransmissions. This can be easily observed on both Linux and Windows platforms by using the **netstat -s** command while Distributed LAN is in progress. Reducing the TCP window size used by Distributed LAN might also help with an oversubscribed network switch. The Windows client **Distributed LAN** tab and the Linux **dpserver** file contain the tuning parameter for the TCP window size. Note that Distributed LAN server remounts are required after changing this parameter.

It is best practice to have all SNFS Distributed LAN clients and servers directly attached to the same network switch. A router between a Distributed LAN client and server could be easily overwhelmed by the data rates required.

It is critical to ensure that **speed/duplex** settings are correct, as this will severely impact performance. Most of the time **auto-detect** is the correct setting. Some managed switches allow setting **speed/duplex**, such as **1000Mb/full**, which disables **auto-detect** and requires the host to be set exactly the same. However, performance is severely impacted if the settings do not match between switch and host. For example, if the switch

is set to **auto-detect** but the host is set to **1000Mb/full**, you will observe a high error rate and extremely poor performance. On Linux the **ethtool** command can be very useful to investigate and adjust **speed/duplex** settings.

In some cases, TCP offload seems to cause problems with Distributed LAN by miscalculating checksums under heavy loads. This is indicated by **bad segments** indicated in the output of **netstat -s**. On Linux, the TCP offload state can be queried by running **ethtool -k**, and modified by running **ethtool -K**. On Windows it is configured through the **Advanced** tab of the configuration properties for a network interface.

The internal bus bandwidth of a Distributed LAN client or server can also place a limit on performance. A basic PCI- or PCI-X-based workstation might not have enough bus bandwidth to run multiple Gigabit Ethernet NICs at full speed; PCI Express is recommended but not required.

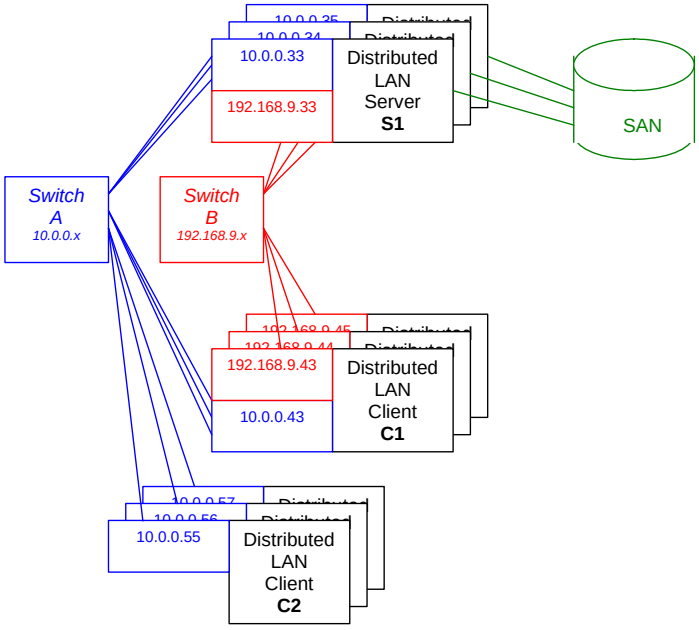
Similarly, the performance characteristics of NICs can vary widely and ultimately limit the performance of Distributed LAN. For example, some NICs might be able to transmit or receive each packet at Gigabit speeds, but not be able to sustain the maximum needed packet rate. An inexpensive 32-bit NIC plugged into a 64-bit PCI-X slot is incapable of fully utilizing the host's bus bandwidth.

It can be useful to use a tool like **netperf** to help verify the performance characteristics of each Distributed LAN network. (When using **netperf**, on a system with multiple NICs, take care to specify the right IP addresses in order to ensure the network being tested is the one you will be running Distributed LAN over. For example, if **netperf -t TCP_RR** reports less than 15,000 transactions per second capacity, a performance penalty might be incurred. Multiple copies of **netperf** can also be run in parallel to determine the performance characteristics of multiple NICs.

Network Configuration and Topology

For maximum throughput, SNFS distributed LAN can utilize multiple NICs on both clients and servers. In order to take advantage of this feature, each of the NICs on a given host must be on a different IP subnetwork. (This is a requirement of TCP/IP routing, not of SNFS - TCP/IP can't utilize multiple NICs on the same subnetwork.) An example of this is shown in the following illustration.

Figure 1 Multi-NIC Hardware and IP Configuration Diagram



In the diagram there are two subnetworks: the blue subnetwork (10.0.0.x) and the red subnetwork (192.168.9.x). Servers such as S1 are connected to both the blue and red subnetworks, and can each provide up to 2 GByte/s of throughput to clients. (The three servers shown would thus provide an aggregate of 6 GByte/s.)

Clients such as C1 are also connected to both the blue and red subnetworks, and can each get up to 2 GByte/s of throughput. Clients such as C2 are connected only to the blue subnetwork, and thus get a maximum of 1 GByte/s of throughput. SNFS automatically load-balances among NICs and servers to maximize throughput for all clients.

Note: The diagram shows separate physical switches used for the two subnetworks. They can, in fact, be the same switch, provided it has sufficient internal bandwidth to handle the aggregate traffic.

Distributed LAN Servers

Distributed LAN Servers must have sufficient memory. When a Distributed LAN Server does not have sufficient memory, its performance in servicing Distributed LAN I/O requests might suffer. In some cases (particularly on Windows,) it might hang.

Refer to the StorNext Release Notes for this release's memory requirements.

Distributed LAN Servers must also have sufficient bus bandwidth. As discussed above, a Distributed LAN Server must have sufficient bus bandwidth to operate the NICs used for Distributed LAN I/O at full speed, while at the same time operating their Fibre Channel HBAs. Thus, Quantum strong recommends using PCI Express for Distributed LAN Servers.

Distributed LAN Client Vs. Legacy Network Attached Storage

StorNext provides support for legacy Network Attached Storage (NAS) protocols, including Network File System (NFS) and Common Internet File System (CIFS).

However, using Distributed LAN Client (DLC) for NAS connectivity provides several compelling advantages in the following areas:

- Performance
- Fault Tolerance
- Load Balancing
- Client Scalability
- Robustness and Stability
- Security Model Consistency

Performance

DLC outperforms NFS and CIFS for single-stream I/O and provides higher aggregate bandwidth. For inferior NFS client implementations, the difference can be more than a factor of two. DLC also makes extremely efficient use of multiple NICs (even for single streams,) whereas legacy NAS protocols allow only a single NIC to be used. In addition, DLC clients communicate directly with StorNext metadata controllers instead of going through an intermediate server, thereby lowering IOP latency.

Fault tolerance

DLC handles faults transparently, where possible. If an I/O is in progress and a NIC fails, the I/O is retried on another NIC (if one is available). If a Distributed LAN Server fails while an I/O is in flight, the I/O is retried on another server (if one is running). When faults occur, applications performing I/O will experience a delay but not an error, and no administrative intervention is required to continue operation. These fault tolerance features are automatic and require no configuration.

Load Balancing

DLC automatically makes use of all available Distributed LAN Servers in an active/active fashion, and evenly spreads I/O across them. If a server goes down or one is added, the load balancing system automatically adjusts to support the new configuration.

Client Scalability

As the following table shows, DLC supports a significantly larger number of clients than legacy NAS protocols:

Largest Tested Configuration			
	NFS	**CIFS**	**DLC**
Number of Clients Tested (via simulation)	4	4	1000

Robustness and Stability

The code path for DLC is simpler, involves fewer file system stacks, and is not integrated with kernel components that constantly change with every operating system release (for example, the Linux NFS code).

Therefore, DLC provides increased stability that is comparable to the StorNext SAN Client.

Consistent Security Model

DLC clients have the same security model as StorNext SAN clients. When CIFS and NFS are used, some security models aren't supported. (For example, Windows ACLs are not accessible when running UNIX Samba servers.)

Windows Memory Requirements

Beginning in version 2.6.1, StorNext includes a number of performance enhancements that enable it to better react to changing customer load. However, these enhancements come with a price: memory requirement.

When running on a 32-bit Windows system that is experiencing memory pressure, the tuning parameters might need adjusting to avoid running the system out of non-paged memory. To determine current operation, open the Task Manager and watch the **Nonpaged** tag in the **Kernel Memory** pane in the lower right hand corner. This value should be kept under 200MB. If the non-paged pool approaches this size on a 32-bit system, instability might occur.

The problem will manifest itself by commands failing, messages being sent to the system log about insufficient memory, the **fsmpm** mysteriously dying, repeated FSM reconnect attempts, and messages being sent to the application log and **cvlog.txt** about socket failures with the status code (10555) which is ENOBUFS.

The solution is to adjust a few parameters on the **Cache Parameters** tab in the SNFS control panel (**cvntclnt**). These parameters control how much memory is consumed by the directory cache, the buffer cache, and the local file cache.

As always, an understanding of the customers' workload aids in determining the correct values. Tuning is not an exact science, and requires some trial-and-error (and the unfortunate reboots) to come up with values that work best in the customer's environment.

The first is the **Directory Cache Size**. The default is 10 (MB). If you do not have large directories, or do not perform lots of directory scans, this number can be reduced to 1 or 2 MB. The impact will be slightly slower directory lookups in directories that are frequently accessed.

Also, in the **Mount Option** panel, you should set the **Paged DirCache** option.

The next parameters control how many file structures are cached on the client. These are controlled by the **Meta-data Cache low water mark**, **Meta-data Cache high water mark** and **Meta-data Cache Max water mark**. Each file structure is represented internally by a data structure called the "cvnode." The cvnode represents all the state about a file or directory. The more cvnodes that there are encached on the client, the fewer trips the client has to make over the wire to contact the FSM.

Each cvnode is approximately 1462 bytes in size and is allocated from the non-paged pool. The **cvnode** cache is periodically purged so that unused entries are freed. The decision to purge the cache is made based on the **Low**, **High**, and **Max** water mark values. The 'Low' default is 1024, the 'High' default is 3072, and the 'Max' default is 4096.

These values should be adjusted so that the cache does not bloat and consume more memory than it should. These values are highly dependent on the customers work load and access patterns. Values of 512 for the **High** water mark will cause the **cvnode** cache to be purged when more than 512 entries are present. The cache will be purged until the low water mark is reached, for example 128. The **Max** water mark is for situations where memory is very tight. The normal purge algorithms takes access time into account when determining a candidate to evict from the cache; in tight memory situations (when there are more than 'max' entries in the cache), these constraints are relaxed so that memory can be released. A value of 1024 in a tight memory situation should work.

Sample FSM Configuration File

This sample configuration file is located in the SNFS install directory under the examples subdirectory named **example.cfg.**

```
# ****************************************************************************
# A global section for defining file system-wide parameters.
#
# For Explanations of Values in this file see the following:
#
# UNIX Users:    man cvfs_config
# Windows Users: Start > Programs > StorNext File System > Help >
# Configuration File Internal Format
#
# ****************************************************************************

GlobalSuperUser         Yes     ## Must be set to Yes for SNMS Managed File
Systems ##
WindowsSecurity         Yes
Quotas                  No
FileLocks               No
DataMigration           No      ## SNMS Managed File Systems Only ##
InodeExpandMin          32K
InodeExpandInc          128K
InodeExpandMax          8M
FsBlockSize             16K
JournalSize             16M
AllocationStrategy      Round
MaxConnections          32
ForceStripeAlignment    Yes
ThreadPoolSize          32      # default 16, 512 KB memory per thread
InodeCacheSize          16K     # 800-1000 bytes each, default 8K

Debug                   0x0
MaxLogSize              16M
MaxLogs                 4

#
# Globals Defaulted
#

# BufferCacheSize               64M     # default 32MB
# StripeAlignSize               2M      # auto alignment threshold, default
```

```
MAX(StripeBreadth)
# MaxMBPerClientReserve       50     # in MBs, default 100 MB reserved per client
# OpHangLimitSecs               300     # default 180 secs
# DataMigrationThreadPoolSize   128     # Managed only, default 8

# ***************************************************************************
# A disktype section for defining disk hardware parameters.
# ***************************************************************************

[DiskType MetaDrive] ##1+1 Raid 1 Mirrored Pair##
Sectors XXXXXXXX     ## Sectors Per Disk From Command "cvlabel -l" ##
SectorSize 512

[DiskType JournalDrive] ##1+1 Raid 1 Mirrored Pair##
Sectors XXXXXXXX
SectorSize 512

[DiskType VideoDrive] ##8+1 Raid 5 Lun for Video##
Sectors XXXXXXXX
SectorSize 512

[DiskType AudioDrive] ##4+1 Raid 3 Lun for Audio##
Sectors XXXXXXXX
SectorSize 512

[DiskType DataDrive] ##4+1 Raid 5 Lun for Regular Data##
Sectors XXXXXXXX
SectorSize 512



# ***************************************************************************
# A disk section for defining disks in the hardware configuration.
# ***************************************************************************

[Disk CvfsDisk0]
Status UP
Type MetaDrive

[Disk CvfsDisk1]
Status UP
Type JournalDrive

[Disk CvfsDisk2]
Status UP
Type VideoDrive

[Disk CvfsDisk3]
Status UP
Type VideoDrive
```

```
[Disk CvfsDisk4]
Status UP
Type VideoDrive

[Disk CvfsDisk5]
Status UP
Type VideoDrive

[Disk CvfsDisk6]
Status UP
Type VideoDrive

[Disk CvfsDisk7]
Status UP
Type VideoDrive

[Disk CvfsDisk8]
Status UP
Type VideoDrive

[Disk CvfsDisk9]
Status UP
Type VideoDrive

[Disk CvfsDisk10]
Status UP
Type AudioDrive

[Disk CvfsDisk11]
Status UP
Type AudioDrive

[Disk CvfsDisk12]
Status UP
Type AudioDrive

[Disk CvfsDisk13]
Status UP
Type AudioDrive

[Disk CvfsDisk14]
Status UP
Type DataDrive

[Disk CvfsDisk15]
Status UP
Type DataDrive

[Disk CvfsDisk16]
Status UP
Type DataDrive
```

```
[Disk CvfsDisk17]
Status UP
Type DataDrive



# ****************************************************************************
# A stripe section for defining stripe groups.
# ****************************************************************************

[StripeGroup MetaFiles]
Status UP
MetaData Yes
Journal No
Exclusive Yes
Read Enabled
Write Enabled
StripeBreadth 256K
MultiPathMethod Rotate
Node CvfsDisk0 0


[StripeGroup JournFiles]
Status UP
Journal Yes
MetaData No
Exclusive Yes
Read Enabled
Write Enabled
StripeBreadth 256K
MultiPathMethod Rotate
Node CvfsDisk1 0


[StripeGroup VideoFiles]
Status UP
Exclusive Yes           ##Exclusive StripeGroup for Video Files Only##
Affinity VideoFiles
Read Enabled
Write Enabled
StripeBreadth 4M
MultiPathMethod Rotate
Node CvfsDisk2 0
Node CvfsDisk3 1
Node CvfsDisk4 2
Node CvfsDisk5 3
Node CvfsDisk6 4
Node CvfsDisk7 5
Node CvfsDisk8 6
Node CvfsDisk9 7
```

```
[StripeGroup AudioFiles]
Status UP
Exclusive Yes           ##Exclusive StripeGroup for Audio File Only##
Affinity AudioFiles
Read Enabled
Write Enabled
StripeBreadth 1M
MultiPathMethod Rotate
Node CvfsDisk10 0
Node CvfsDisk11 1
Node CvfsDisk12 2
Node CvfsDisk13 3


[StripeGroup RegularFiles]
Status UP
Exclusive No            ##Non-Exclusive StripeGroup for all Files##
Read Enabled
Write Enabled
StripeBreadth 256K
MultiPathMethod Rotate
Node CvfsDisk14 0
Node CvfsDisk15 1
Node CvfsDisk16 2
Node CvfsDisk17 3
```